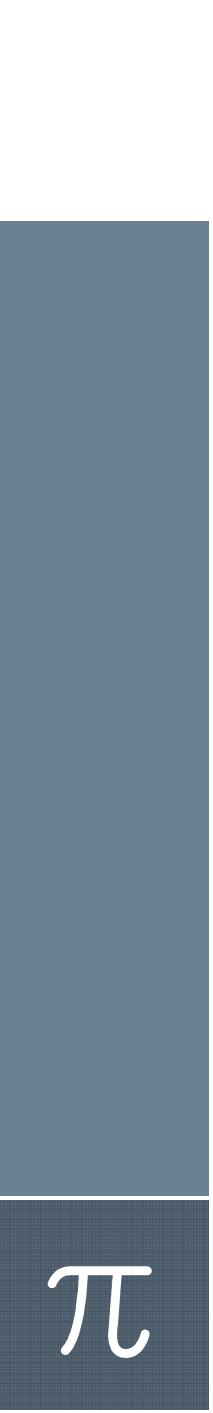
Embedded Thursday

Variables + Timers + PWM

Ivan Quiroz



Today

- The project Goal and System
- Recap
- Variables
- Hardware PWM
- Timers
- Intro to Interrupts

Goal Description

- Learn C as embedded language
- Use C to understand underlying processor
- Have a project so learning stays
  - We are making a self balancing robot

Todays Goal

- Learn to set PWM
- Learn to set Timers

Recap on Inputs/Outputs and registers

- Step 1: enable ports use register RCC_AHB1ENR
- Step 2: set Ports as IN or OUT writing to register GPIOx_MODER
- Step 3: set Pin HIGH or LOW writing to register GPIOx_ODR
- Step 4: read Input Pin by reading register GPIOx_IDR
- Debounce if you are reading a switch

Operators

- XOR `^=` Use: switch bit to opposite value
- OR `|=` Use: Impact a bit, don't disturb others by OR'ing desired bit with 0x001
- AND `&=` Use: Impact a bit, don't disturb others by OR'ing desired bit with 0x110
- AND `&` Use: Mask a bit with using 0x001

Variables brief Introduction

Variable allocate a location in memory during compiling

The datatype defines the expected data we will use in a variable

```
#include <stdint.h>
```

- `int8_t`
- `int16_t`
- `int32_t`
- `uint8_t`
- `uint16_t`
- `uint32_t`

Data type	Precision	Range
unsigned char	8-bit unsigned	0 to +255
signed char	8-bit signed	-128 to +127
unsigned int	compiler-dependent	
int	compiler-dependent	
unsigned short	16-bit unsigned	0 to +65535
short	16-bit signed	-32768 to +32767
unsigned long	unsigned 32-bit	0 to 4294967295L
long	signed 32-bit	-2147483648L to 2147483647L
float	32-bit float	±10-38 to ±10+38
double	64-bit float	±10-308 to ±10+308

`Volatile:` A variable that may change at any time without any action being taken by the code

```
volatile int8_t Switchstatus
```

In embedded volatile is used to

- Define I/O ports (value of ports can change outside of software action. i.e. switch pressed
- Share a global variable between the main program and an interrupt service routine.
- Global variables accessed by multiple tasks within a multi-threaded application

Variables

```
int main(void)
{
//  RCC->AHB1ENR |= RCC_AHB1ENR_GPIODEN; // enable
    RCC->AHB1ENR |= 0x00000008;      // enable the
    RCC->AHB1ENR |= 0x00000001;      // enable the

    GPIOD->MODER |=  0X55000000;     // Set Port-D
    GPIOA->MODER &=  0XFFFFFFFE;       // Set Por
/*  GPIOD->MODER |= (1 << 24);       // another way

    int8_t i;
    volatile int SwitchStatus;

    GPIOD->ODR = 0x0000;                        //

    while (1){
/*      GPIOD->ODR ^= (1 << 12);     // another way


        SwitchStatus = ((GPIOA->IDR & 0x1) == 0);

        if (!SwitchStatus){
//          GPIOD->ODR ^= 0b1010000000000000;   //
            GPIOD->ODR ^= 0xD000;               //
//          GPIOD->ODR |= 0xF000;
            for (i = 0; i < 500000; i++);       //
        }
    }

}
```

Table 1. STM32F411xC/E register boundary addresses

Boundary address	Peripheral	Bus	Register map
0x5000 0000 - 0x5003 FFFF	USB OTG FS	AHB2	*Section 22.16.6: OTG_FS register map on page 744*
0x4002 6400 - 0x4002 67FF	DMA2		*Section 9.5.11: DMA register map on page 194*
0x4002 6000 - 0x4002 63FF	DMA1		
0x4002 3C00 - 0x4002 3FFF	Flash interface register		*Section 3.8: Flash interface registers on page 58*
0x4002 3800 - 0x4002 3BFF	RCC		*Section 6.3.22: RCC register map on page 133*
0x4002 3000 - 0x4002 33FF	CRC		*Section 4.4.4: CRC register map on page 68*

```
#define RCC   ((RCC_TypeDef *) RCC_BASE)
```

Pointer Definition

0x400023800

```
#define PERIPH_BASE          ((uint32_t) 0x40000000U)
#define AHB1PERIPH_BASE      (PERIPH_BASE + 0x00020000U)
#define RCC_BASE             (AHB1PERIPH_BASE + 0x3800U)
```

Table 21. RCC register map and reset values for STM32F411xC/E

Addr. offset	Register name	31	30	29	28	27	26	25	24	23	22	21	20	19	18	17	16	15	14	13	12	11	10	9	8	7	6	5	4	3	2	1	0
0x30	RCC_AHB1ENR	Reserved									DMA2E	DMA1E	Reserved								CRCEN	Reserved				GPIOHE	Reserve		GPIOEE	GPIODE	GPIOCE	GPIOBE	GPIOAE

Timers – TIM4

- A timer is a special register that once enabled it counts
  - The bucket to count is only so big
  - Once the bucket is full, it overflows
  - You can prefill the bucket
  - You can set the speed to fill the bucket
  - Interrupts can inform you if bucket has overflown
- We will use the Advance Control Timer TIM4
  - 16 Bit bucket 2^{16} :: 0 to 65,536 (count up/down)
  - Once it reaches value on Auto-Reload Register it restarts
  - We will use it for PWM generation (square wave form)
  - Use pre-scalars to set speed of count

STM32F411E-DISCO

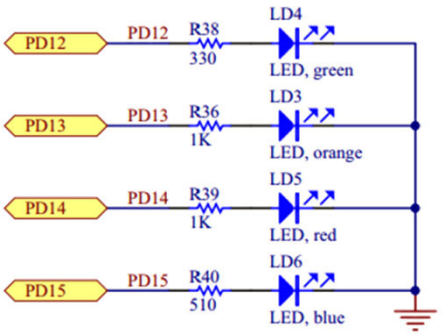


Table 9. Alternate function ma

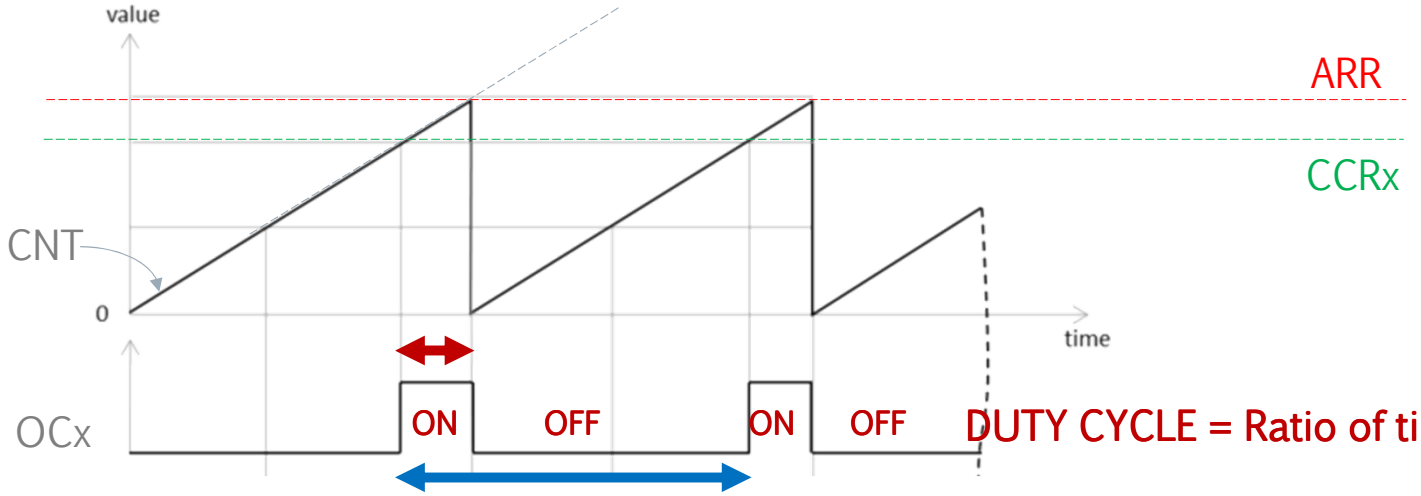
Port		AF00	AF01	AF02	AF03	AF04	AF05	AF06
		SYS_AF	TIM1/TIM2	TIM3/ TIM4/ TIM5	TIM9/ TIM10/ TIM11	I2C1/I2C2/ I2C3	SPI1/I2S1SPI2/ I2S2/SPI3/ I2S3	SPI2/I2S2/ SPI3/ I2S3/SPI4/ I2S4/SPI5/ I2S5
Port D	PD12	-	-	TIM4_CH1	-	-	-	-
	PD13	-	-	TIM4_CH2	-	-	-	-
	PD14	-	-	TIM4_CH3	-	-	-	-
	PD15	-	-	TIM4_CH4	-	-	-	-

Given location of LED we will use Timer 4 (TIM4) to generate PWM

- Port-D.Pin12: AF02 - CH1
- Port-D.Pin13: AF02 - CH2
- Port-D.Pin14: AF02 – CH3
- Port-D.Pin15: AF02 – CH4
- *Source: STM32F411 datasheet table-9*

PWM Mode (Reference Manual 13.3.9)

- Generate a square signal of determined frequency
  - Frequency determined by TIM4_ARR register
  - Duty Cycle determined by TIM4_CCR1 register



PWM Setup (follow section 13.3.9)

a. Configure ports as outputs
   a. Enable clock to Port D
   b. Set PD12-15 as alternate function
   c. Enable clock to Timer4 bus
   d. Connect Timer output to PD12-15 actual pins

b. Set up timer TIM4
   a. Count upwards
   b. Enable PWM
   c. Set clock divider
   d. Set prescalar

c. Configure timer for duty cycle

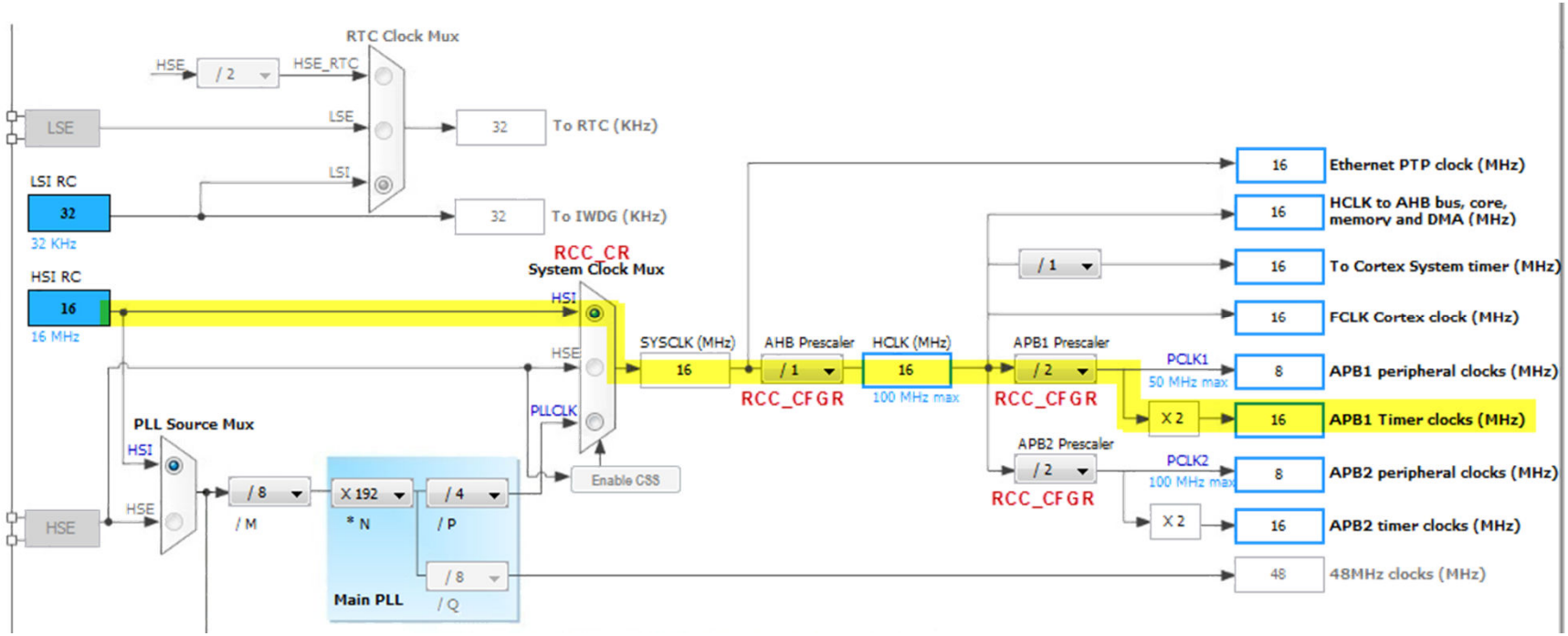
TIM4 registers to use:

- ❑ _CCMR1
- ❑ _CCRx
- ❑ _PSC
- ❑ _ARR
- ❑ _CCER
- ❑ _CR1

- ❑ _CNT

Clock Source

- STM32F411E-DISCO has a 8Mhz input clock
  - SystemInit() sets Clock source as HIS (Internal 16MHz clock)
  - SystemInit() sets APB1 clock for timers = 16MHz



PWM Frequency and Duty Cycle

- Timer4 Counter Clock Frequency = 16Mhz (CL_CNT)
- TIM4_PSC prescaler = CL_CNT/2 = 8Mhz
- PWM Frequency = CNT_CK / ARR + 1
  - I want a 1.6KHz frequency (pulled out of thin air)
  - If ARR = 4999 Pwm= 8,000Khz/(4999+1)= 1.6Khz
- Duty Cycle:
  - ARR+1: Controls frequency = 4999 → 100%
  - CCRx: Controls ON time = X → 50%
    - X = ((4999 +1)* 50%)/100% = 2500

PWM Code

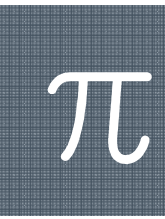
```
#include "stm32f4xx.h"

void PWM_PortInit()

{
 RCC->AHB1ENR |= 0x00000008;  // enable Clock to the PORT-D
 RCC->APB1ENR |= 0x00000004;  // Enable clock to TIM4 timer
 GPIOD->MODER = 0xAA000000;   // Set Port-D pin12 to 14 to alternate function OUTPUTS
 GPIOD->OSPEEDR = 0xAA000000; // set port speed to fast for ports D12-14 (50Mhz)
 GPIOD->AFR[1] |= 0x22220000; //Set alt. func. PD12:(TIM4-CH1), PD13:(TIM4-CH2), PD14:(TIM4-CH3), PD15:(TIM4-CH4)
}

void PWM_TimerInit()
{
 TIM4->CCMR1= 0x6868;  // Enable OC1-2 for PWM use on Mode 1
 TIM4->CCMR2 = 0x6868; // Enable OC3-4 for PWM use on Mode 1

 TIM4->CCR1= 0x834;   // Duty Cycle= 50%
 TIM4->CCR2= 0x834;   // Duty Cycle
 TIM4->CCR3= 0x834;   // Duty Cycle
 TIM4->CCR4= 0x834;   // Duty Cycle

 TIM4->PSC= 0x0001;  // APB1 from HSI = 16Mhz, scaled by 2 by SMCR. Further scaled down by PSC by 2 to 8Mhz
 TIM4->ARR= 4999;    // Computed by 8Mhz/(4199+1). We want 1.6KHz = 8Mhz / (ARR + 1) solve for ARR = 4999

TIM4->CCER |= 0x1111;              // Enable CH1-4 output
TIM4->CR1 |= 0x0081; // Divide clock by 1, auto-reload, and start timer
}
```

Code - Continue

```
int main(void)
{
 SystemInit();
 RCC->AHB1ENR |= 0x00000001;     // enable the PORT-A
 GPIOA->MODER &= 0xFFFFFFFD;     // Set Port-A-Pin0 as input
 PWM_PortInit();                 // Initialize port for Timer set up
 PWM_TimerInit();                // Initilize PWM Timer4

 int32_t i;
 int32_tDutyCycle;
 volatile int8_t SwitchStatus;

 while (1){

  SwitchStatus = ((GPIOA->IDR & 0x1) == 0);// Read status of Port-A_Pin0 (masking bit0 by AND to 1)

  if (!SwitchStatus){
   DutyCycle = DutyCycle + 100;
   TIM4->CCR1 = DutyCycle;
   TIM4->CCR2 = DutyCycle;
   TIM4->CCR3 = DutyCycle;
   TIM4->CCR4 = DutyCycle;
   for (i = 0; i < 500000; i++);// Add delay by wasting time adding 1 to i from 0 to 500K
  }
 }
}
```

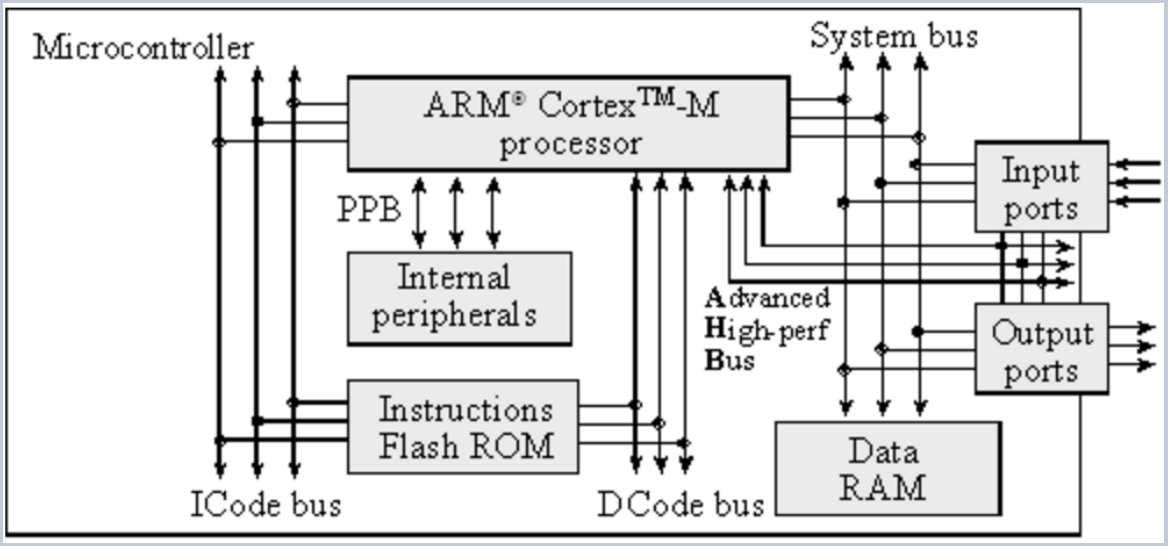
Back Up Slides

Hardware Reference Material



SIMPLIFIED STM34F411 ARCHITECTURE

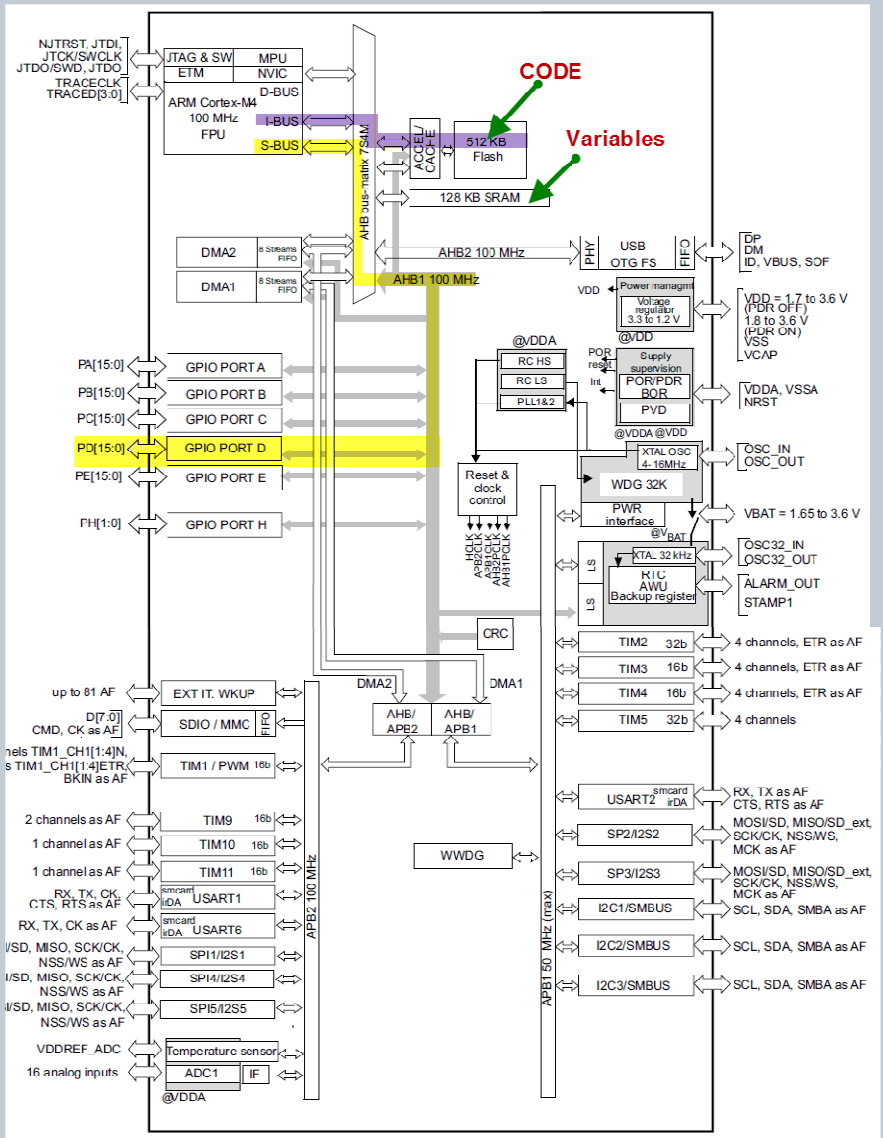
- I-Code Bus use to fetch instructions from Flash ROM
- System Bus: use to work with variables and IO Ports
- D-Code Bus: debug bus
- Adv Hi Bus: Connection to IO ports and dedicated USB ports



STM32F411 BLOCKDIAGRAM

Note the following buses:

- **RCC->AHB1ENR** needed as Port D uses AHB1 (yellow)



PWM DIAGRAM

CLOCK GENERATION

- APB1 clock depends on RCC_CR settings
- APB1 CLK is x2 for TIMx
- PSC can divide by 2
- ARR sets max count
- CCR set max ON time